Sentence Classification Practices

using

Logistic Regression

Task - 1

Ibrahim R. Hallac

April 15, 2017

0.1 Definition

Sentiment classification on the referenced dataset of the paper.

DATASET	
Content	Movie reviews - collected from IMDB
Extension	txt
Size	2000 files
Classes	1000 positive 1000 negative

Ref: Movie review sentiment (MR) (Pang and Lee, 2005)

0.2 Steps

1. **Create a dictionary object of the glove model**. This is performed for each glove data.
   I implemented a function ***getGloveVec*** which

   - takes dimension size of the preferred glove model
   - goes to the directory where glove model data are located
   - returns a dictionary object with:
     word as **key**
     vector value for that word as **value**

   glove dictionary looks like this:

   $$\text{glove}[\text{"}ibrahim\text{"}] \quad = \quad [-0.123 \quad 0.296 \quad 0.43217 \quad ... \quad -0.7964]$$

2. **Create training and test dataset matrices**. This is done automatically by:

   - going to dataset folder and using folder names as labels
   - under each folder, iterate over the files
   - clean the text from stopwords (stopword list taken from nltk library)
   - create glove word model representation of the text by:
     - go through the tokens in the file
     - sum up the glove equivalents of the tokens
     - divide the sum with number of tokens
     - this is our average glove vector for the given text!
     - add glove vector and it's label to dataset matrix
     - repeat this steps for all of the files
   - Since the files are read one by one, dataset matrix is too homogeneous, shuffle it.
   - Split the dataset into train and test matrices as %80 and %20 of the all data, respectively

3. **Fit initial θ parameters**

   Logistic regression hypothesis *is defined as:*

   $$h_\theta(x) = g(z) \Rightarrow z = \theta^T X \tag{1}$$

   g(z) is the **Sigmoid Function** which is calculated as:

   $$g(z) = \frac{1}{1+e^-z} g(\theta^T X) = \frac{1}{1+e^-\theta^T X} \tag{2}$$

4. **Predict !**

0.3 Learning Curve

```
def learningCurve(self):
        sizeAcc = {}
        (m, n) = self.Train_X.shape
        # create a list of training size according to initial-
        #-split of training data
        # for 1600 of training data
        # 200, 400, 800, 1200, 1600
        t_sizes = [int(m/8), int(m/4), int(m/2), int(m*3/4), m]
        thetas = np.zeros(n)
        # find theatas for each size of training set
        for t_size in t_sizes:
            trained = optimizer.fmin_tnc(func=self.costFunction,
                                         x0=thetas,
                                         fprime=self.gradient,
                                         args=(self.Train_X[0:t_size],
                                               self.Train_y[0:t_size]),
                                         disp=False)
            theta_min = np.matrix(trained[0])
            # key, value
            # (training_size, theatas [])
            sizeAcc[t_size] = self.test(theta_min)
        return sizeAcc
```

0.4 Main Program

```
def main():
    mainPath = os.getcwd()
    dataPath = os.path.join(mainPath, "review_polarity")
    resultPath = os.path.join("results")
    dimSizes = ["50", "100", "200", "300"]
    createLearningCurve(dataPath, dimSizes)
    loadLearningCurve(resultPath, dimSizes)

def createLearningCurve(path, dimSizes):
    for dim in dimSizes:
        sizeAccAvr = {}
        # dataset is shuffled everytime
        # i think it is a good idea to do the tests for 10 times
        # and use the average accuracy as resulting learning rate
        for i in range(10):
            lr_c = lr.LogisticRegression(data_path=path, dim=dim)
            sizeAcc = lr_c.learningCurve()
            if len(sizeAccAvr.keys()) is 0:
                for key in sizeAcc.keys():
                    sizeAccAvr[key] = sizeAcc[key]
            else:
                for key in sizeAcc.keys():
                    sizeAccAvr[key] += sizeAcc[key]
        for key in sizeAccAvr.keys():
            sizeAccAvr[key] = sizeAccAvr[key]/10
        # serialize learning rates for each glove vector dimention-
        # using pickle
        fw = open(os.path.join("results",
                               "sizeAccAvr_{}.pkl".format(dim)), "w")
        pickle.dump(sizeAccAvr, fw)
        fw.close()

def loadLearningCurve(path, dimSizes):
    # plot all of the learning curves that are saved with pickle
    for dim in dimSizes:
        fr = open(os.path.join(path, "sizeAccAvr_{}.pkl".format(dim)), "r")
        sizeAccc = pickle.load(fr)
        fr.close()
        od = collections.OrderedDict(sorted(sizeAccc.items()))
        plotLearningCurve(od, dim)
```

0.5 Results

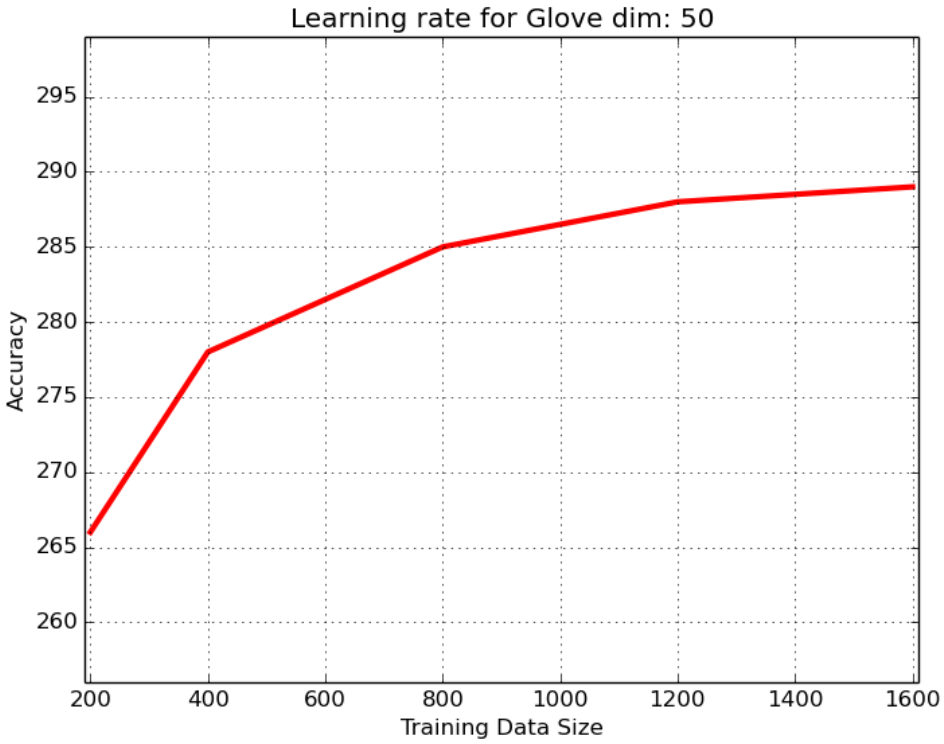

Figure 1: glove-50d

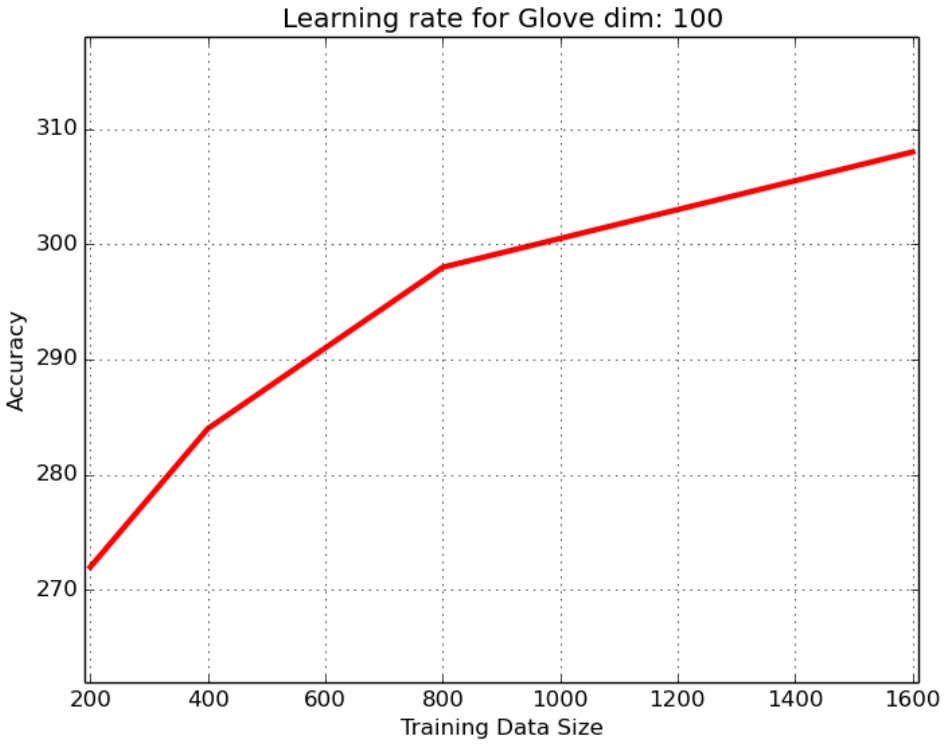

Figure 2: glove-100d

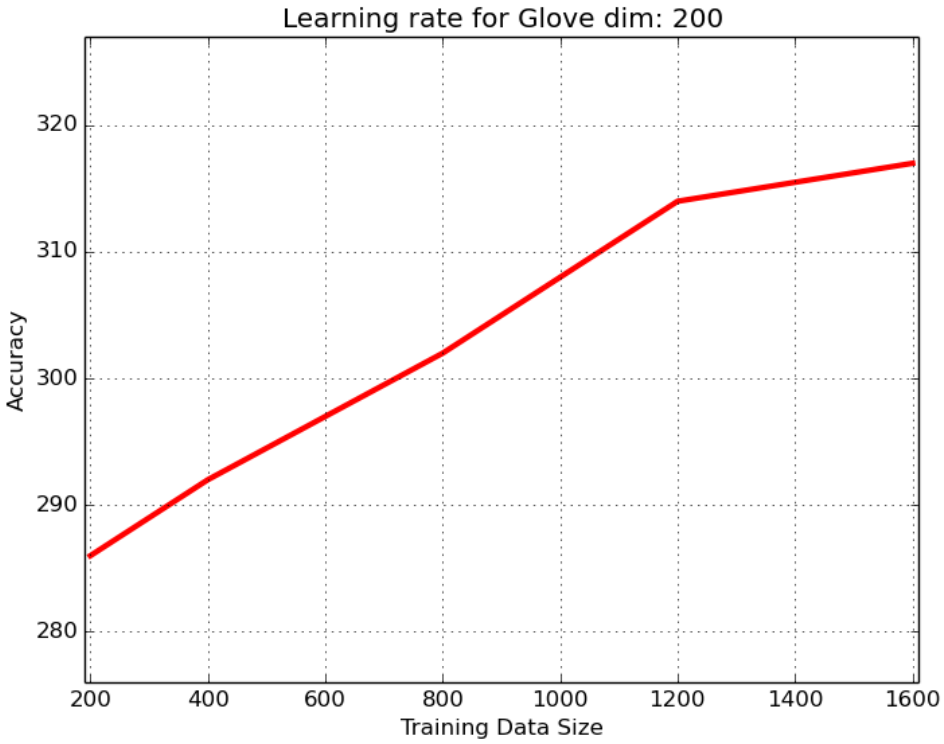

Figure 3: glove-200d

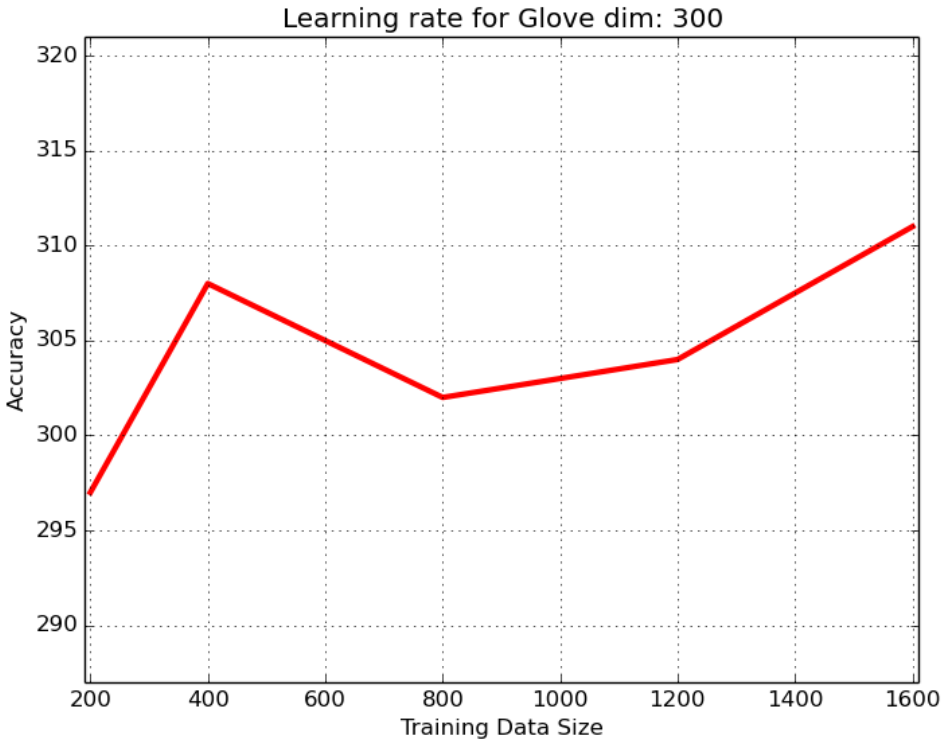

Figure 4: glove-300d